Chapter 12. Web Services

J2EE 1.4 의 큰 특징은 J2EE components 를 웹서비스 제공자와 수요자 둘다의 기능을 가지게 한다는 것이다. J2EE application 은 SLSB 을 이용하여 EJB 티어 나 POJO 를 이용하여 web 티어를 를 웹서비스로 가능하게 해준다. 또한 J2EE components 가 외부의 웹서비스로 참조될수 있도록 할 수있는 표준적인 방법을 포함하고 있다.

12.1. JAX-RPC Service Endpoints

JAX-RPC service endpoints (JSEs) 는 web 티어레벨의 웹서비스를 제공한다. 간단한 java object 를 Servlets 처럼 작동하도록 하는 형식을 취하고 있다. 우리는 여기서 hello 웹서비스를 구현해보도록 하겠다.

```
package org.jboss.hello;

public class HelloPojo
{
    public String hello(String name)
    {
        return "Hello " + name + "!";
    }
}
```

HelloPojo 에 대한 특별한 사항은 없다. 여기서는 특정한 인터페이스를 구현하고나 하거나 비즈니스 메소드 외에 따로이 추가되는 메소드가 없다.
hello 메소드는 웹서비스 로 사용되기 위해 만들어진 메소드라고 할 수 있다. 여기서는 방갑게 맞이해주는 부분 외에 따로 보여주는 것은 없다.

HelloPojo

추가적으로 service endpoint interface(SEI) 를 정의해야 한다. 이것은 웹서비스의 인터페이스를 정의해주는 역할을 한다.

```
package org.jboss.ws.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello
```

```
public interface Hello
    extends Remote
{
    public String hello(String name)
        throws RemoteException;
}
```

SEI는 Remote 상속받고 메소드는 반드시 RemoteException 을 던지도록 선언해야 한다. 상당에 보면 interface의 간단하게 표현되는 것을 볼 수 있다.
Hello 를 디플로이하려면 몇가지 추가적인 Deployment descriptor 가 필요하다.

JSE 는 servlet 과 닮은 점이 전혀 비록 없지만 JSE 를 **web.xml** 파일에 Servlet 과 같이 매핑해줄 수 있다. servlet 처럼 웹서비스 를 구현한 클래스를 선언해줄 필요가 있다 아래 hello web service를 디플로이시 필요한 정의를 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                             http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">

    <servlet>
        <servlet-name>HelloWorldWS</servlet-name>
        <servlet-class>org.jboss.ws.hello.HelloPojo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldWS</servlet-name>
        <url-pattern>/Hello</url-pattern>
    </servlet-mapping>

</web-app>
```

servlet mapping 에서의 URL 패턴은 외부에서 보이는 부분에 대한 설정을 뜻할 뿐이다. 이것을 통해 web service 를 제공하는 URL 을 조작할 수 있다.

web.xml 파일은 web service 관련된 모든 설정을 가지고 있지 않다. webserices.xml 은 JBoss 에게 web service 를 일반적인 servlet 으로가 아닌 web service를 수행하는 servlet 이라는 것을 알려준다. 또한 WSDL 파일 과 JAX-RPC mapping 파일이 추가적으로 필요하다. 이 파일들은 wscompile을 이용하여 생성할 수 있다. (WSDP)

```
wscompile -classpath <classpath> -gen:server -f:rpcliteral -mapping mapping.xml config.xml
```

WSDL 파일과 JAX-RPC mapping 파일을 생성하기 위해서는 **config.xml** 파일이 필요한다. 아래에 config.xml 파일을 샘플로 보여주고 있다.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
    <service name="HelloService"
             targetNamespace="http://hello.ws.jboss.org/"
             typeNamespace="http://hello.ws.jboss.org/types"
             packageName="org.jboss.ws.hello">
        <interface name="org.jboss.ws.hello.Hello"/>
    </service>
</configuration>
```

service element는 web service 에 정의하는데 사용된다. attributes 에 대해 알아보자.

- **name**: This is the name of the web service.
- **targetNamespace**: Web services require namespaces just like Java classes do. It's a common practice to use a URL namespace that corresponds to the Java namespace given.
- **typeNamespace**: This specifies the namespace to use for custom types.
- **packageName**: This is the base package name that your web services classes live under.

config.xml 참조하여 **wscompile**을 돌리면 WSDL 파일이 생성된다. 하지만 **WSDL** 파일에 SOAP address 가 비어있는 것을 확인 할 수 있다. JBoss 는 웹서비스로 디플로이 될때 WSDL 의 정확한 위치를 넣어준다.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
    targetNamespace="http://hello.ws.jboss.org/"
    xmlns:tns="http://hello.ws.jboss.org/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types/>
    <message name="Hello_hello">
        <part name="String_1" type="xsd:string"/>
    </message>
    <message name="Hello_helloResponse">
        <part name="result" type="xsd:string"/>
```

```
    </message>
    <portType name="Hello">
        <operation name="hello" parameterOrder="String_1">
            <input message="tns:Hello_hello"/>
            <output message="tns:Hello_helloResponse"/>
        </operation>
    </portType>
    <binding name="HelloBinding" type="tns:Hello">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
        <operation name="hello">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal" namespace="http://hello.ws.jboss.org/"/>
            </input>
            <output>
                <soap:body use="literal" namespace="http://hello.ws.jboss.org/"/>
            </output>
        </operation>
    </binding>
    <service name="HelloService">
        <port name="HelloPort" binding="tns:HelloBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
    </service>
</definitions>
```

우리는 또한 **JAX-RPC mapping 파일**을 볼 수 있따.

```
<?xml version="1.0" encoding="UTF-8"?>
<java-wsdl-mapping version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                        http://www.ibm.com/webservices/xsd/j2ee_jaxrpc_mapping_1_1.xsd">
    <package-mapping>
        <package-type>org.jboss.ws.hello</package-type>
        <namespaceURI>http://hello.ws.jboss.org/types</namespaceURI>
    </package-mapping>
    <package-mapping>
        <package-type>org.jboss.ws.hello</package-type>
        <namespaceURI>http://hello.ws.jboss.org/</namespaceURI>
    </package-mapping>
    <service-interface-mapping>
        <service-interface>org.jboss.ws.hello.HelloService</service-interface>
        <wsdl-service-name xmlns:serviceNS="http://hello.ws.jboss.org/">
```

```
        <wsdl-service name xmlns:serviceNS= http://hello.ws.jboss.org/ >
            serviceNS:HelloService
        </wsdl-service-name>
        <port-mapping>
            <port-name>HelloPort</port-name>
            <java-port-name>HelloPort</java-port-name>
        </port-mapping>
    </service-interface-mapping>
    <service-endpoint-interface-mapping>
        <service-endpoint-interface>org.jboss.ws.hello.Hello</service-endpoint-interface>
        <wsdl-port-type xmlns:portTypeNS="http://hello.ws.jboss.org/">
            portTypeNS:Hello
        </wsdl-port-type>
        <wsdl-binding xmlns:bindingNS="http://hello.ws.jboss.org/">
            bindingNS:HelloBinding
        </wsdl-binding>
        <service-endpoint-method-mapping>
            <java-method-name>hello</java-method-name>
            <wsdl-operation>hello</wsdl-operation>
            <method-param-parts-mapping>
                <param-position>0</param-position>
                <param-type>java.lang.String</param-type>
                <wsdl-message-mapping>
                    <wsdl-message xmlns:wsdlMsgNS="http://hello.ws.jboss.org/">
                        wsdlMsgNS:Hello_hello
                    </wsdl-message>
                    <wsdl-message-part-name>String_1</wsdl-message-part-name>
                    <parameter-mode>IN</parameter-mode>
                </wsdl-message-mapping>
            </method-param-parts-mapping>
            <wsdl-return-value-mapping>
                <method-return-value>java.lang.String</method-return-value>
                <wsdl-message xmlns:wsdlMsgNS="http://hello.ws.jboss.org/">
                    wsdlMsgNS:Hello_helloRespon
                </wsdl-message>
                <wsdl-message-part-name>result</wsdl-message-part-name>
            </wsdl-return-value-mapping>
        </service-endpoint-method-mapping>
    </service-endpoint-interface-mapping>
        </java-wsdl-mapping>
```

webservices.xml 파일의 내용을 보면 wsdl 파일과의 링크 부분을 보여주고 있으며 또한 jaxrpc-mapping 파일 부분도 링크되어져 있다.또한 추가적으로 port-component element 에서는 WSDL 의 port 부분을 구현된 특정서비스를 매핑하고 있다. 또한 JSE 를 보면 servlet-link 에서 web.xml 파일에서 정의되어 있는 Servlet 명을 servlet-link 에 명시해준다.

```
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                        http://www.ibm.com/webservices/xsd/j2ee_web_services_1_1.xsd"
    version="1.1">
    <webservice-description>
        <webservice-description-name>HelloService</webservice-description-name>
        <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
        <jaxrpc-mapping-file>WEB-INF/mapping.xml</jaxrpc-mapping-file>
        <port-component>
            <port-component-name>Hello</port-component-name>
            <wsdl-port>HelloPort</wsdl-port>
            <service-endpoint-interface>
                org.jboss.ws.hello.Hello
            </service-endpoint-interface>
            <service-impl-bean>
                <servlet-link>HelloWorldWS</servlet-link>
            </service-impl-bean>
        </port-component>
    </webservice-description>
</webservices>
```

web service 를 포함한 WAR 파일을 디렉토리 구조로 봤을 때 아래와 같은 그림으로 보이게 된다. 모든 deployment descriptors들은 WEB-INF 밑에 있다.
wsdl 파일은 wsdl 서브디렉토리에 있는 것에 주의 할 필요가 있다.(Figure 12.1, “The structure of hello-servlet.war”.)

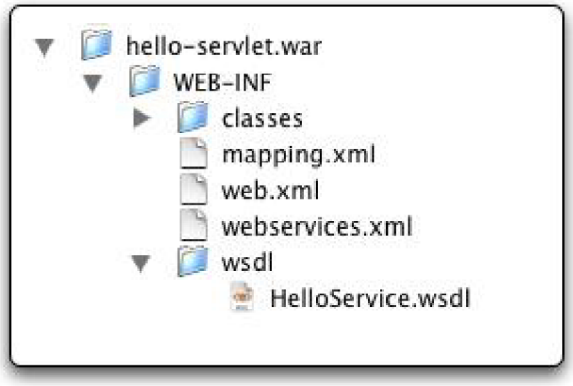

Figure 12.1. The structure of hello-servlet.war

hello web service 를 디플로이하고 테스트하기 위해서는 examples 디렉토리에서 아래와 같이 실행하면 된다.

```
[examples]$ ant -Dchap=ws -Dex=1 run-example
...
run-example1:
     [echo] Waiting for 5 seconds for deploy...
     [java] Contacting webservice at http://localhost:8080/hello-servlet/Hello?wsdl
     [java] hello.hello(JBoss user)
     [java] output:Hello JBoss user!
```

server log 에서 생성된 WSDL 과 wsdd 파일의 임시위치를 포함한 디플로이 정보를 볼 수 있다. 또한 web service url 을 볼 수 있다.

URL 을 통해 JBoss에서 제공하는 WSDL 파일을 볼 수 있다. 웹 어플리케이션 명이 hello-servlet 이고 우리가 web.xml 파일에서 servlet을 매핑한 /Hello 했었다. 따라서 /hello-servlet/Hello 로 web service가 매핑되었음을 확인할 수 있다. ?wsdl 을 붙여줌으로 WSDL 파일을 리턴 받을 수 있다.

WSDL 파일에 매핑된 URL 이 정확하지 않다면 JBoss에서 제공되는 system 쪽의 web services 를 통해 /ws4ee/services 로 확인할 수 있다.

Figure 12.2. The web services list

서비스 리스트는 현재 web services로 등록된 모든 것을 보여준다.

12.2. EJB Endpoints

Web services 는 EJB 티어도 제공할 수 있다. SLSB 는 JAX-RPC endpoint 와 같은 방법으로 web service의 endpoint 로 서비스 할 수 있다.
이작업을 하기위해서는 session bean 에서 HelloServet 예제를 적용하게 될 것이다.

```
package org.jboss.ws.hello;

import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class HelloBean
    implements SessionBean
{
    public String hello(String name)
    {
        return "Hello " + name + "!";
    }

    public void ejbCreate() {};
    public void ejbRemove() {};

    public void ejbActivate() {}
    public void ejbPassivate() {}

    public void setSessionContext(SessionContext ctx) {}
}
```

여기 보이는 session bean은 아주 간단하다. session bean 은 보통 home interface와 local interface 또는 remote interface 가 필요하다. 그러나 web service endpoint 로 제공되는 session bean 은 interface 파일들을 생략할 수 있다. 그러나 우리는 JSE 예제에서 사용되었던 hello service endpoint 는 반드시 필요하다.

ejb-jar.xml 파일은 session bean의 표준이다. 보통의 session bean parameters에 대한 설명은 5장 EJBs on JBoss에서 설명하고 있다.
여기서 집중해야 하는 것은 service-endpoint element 이다. 이 element는 web service를 위한 service endpoint interface로 이용된다.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
    <display-name>chapter 12 EJB JAR</display-name>
    <enterprise-beans>
        <session>
            <ejb-name>HelloBean</ejb-name>
            <service-endpoint>org.jboss.ws.hello.Hello</service-endpoint>
```

```
            <service-endpoint>org.jboss.ws.hello.Hello</service-endpoint>
            <ejb-class>org.jboss.ws.hello.HelloBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <method-permission>
            <unchecked/>
            <method>
                <ejb-name>HelloBean</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <container-transaction>
            <method>
                <ejb-name>HelloBean</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

webservices.xml 파일도 함께 반드시 필요로 한다. 해당 파일은 아래에서 보여주고 있다. WAR 파일을 위한 부분을 확인할 수 있다.

```
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://www.ibm.com/webservices/xsd/j2ee_web_services_1_1₩.xsd" version="1.1">
    <webservice-description>
        <webservice-description-name>HelloService</webservice-description-name>

        <wsdl-file>META-INF/wsdl/HelloService.wsdl</wsdl-file>
        <jaxrpc-mapping-file>META-INF/mapping.xml</jaxrpc-mapping-file>

        <port-component>
            <port-component-name>Hello</port-component-name>
            <wsdl-port>HelloPort</wsdl-port>
            <service-endpoint-interface>org.jboss.ws.hello.Hello</service-endpoint-interface>
            <service-impl-bean>
                <ejb-link>HelloBean</ejb-link>
            </service-impl-bean>
        </port-component>
    </webservice-description>
```

```
</webservices>
```

첫번째 틀린점은 WEB-INF/wsdl 디렉토리 대신하여 META-INF/wsdl 디렉토리 밑에 WSDL 파일이 있다는 것이다.
두번째 틀린점은 service-impl-baen element에 ejb-link 가 session bean 의 ejb-name을 가리키고 있다는 점이다.
WSDL 파일과 JAX-RPC mapping 파일은 전예제와 다르지 않다.

어플리케이션을 패키징하고 디플로이 하기위해서는 아래와 같이 실행하면 된다.

```
[examples]$ ant -Dchap=ws -Dex=2 run-example
...
run-example2:
     [copy] Copying 1 file to /tmp/jboss-4.0.3/server/default/deploy
     [echo] Waiting for 5 seconds for deploy...
     [java] Contacting webservice at http://localhost:8080/hello-ejb/Hello?wsdl
     [java] hello.hello(JBoss user)
     [java] output:Hello JBoss user!
```

The test program run here is the same as with the servlet example, except that we use a different URL for the WSDL. JBoss composes the WSDL using the base name of the EJB JAR file and the name of the service interface. However, as with all web services in JBoss, you can use the `http://localhost:8080/ws4ee/services` service view shown in Figure 12.2, "The web services list" to verify the deployed URL of the WSDL.

테스트 프로그램은 servlet 샘플과 같은 방식으로 동작한다. 또한 JBoss에서 제공되는 system 쪽의 web services 를 통해 /ws4ee/services 로 확인할 수 있다.

12.3. Web Services Clients

이제 web service를 이용하는 부분을 만들어보자 쉽게 클라이언트를 만들어보자

12.3.1. A JAX-RPC client

전체 JAX-RPC 프로그래밍 모델은 J2EE applications 과 clients 이다. 우리는 client programming 기술에 대한 전반적인 부분을 설명하지는 않을 것이다.
그러나 클라이언트에서 dynamic proxy invocation mechanism 에 대해서는 예를 들어 보겠다.

```
package org.jboss.ws.client;

import org.jboss.ws.hello.Hello;

import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;

import javax.xml.namespace.QName;

import java.net.URL;

public class HelloClient
{
    public static void main(String[] args)
        throws Exception
    {
        String urlstr   = args[0];
        String argument = args[1];

        System.out.println("Contacting webservice at " + urlstr);

        URL url =  new URL(urlstr);

        QName qname = new QName("http://hello.ws.jboss.org/",
                                "HelloService");

        ServiceFactory factory = ServiceFactory.newInstance();
        Service        service = factory.createService(url, qname);

        Hello          hello   = (Hello) service.getPort(Hello.class);

        System.out.println("hello.hello(" + argument + ")");
        System.out.println("output:" + hello.hello(argument));
    }
}
```

JAX-RPC 클라이언트는 Hello service endpoint interface 를 이용한다. dynamic proxy 를 생성하여 command line 에서 들어온 argument 가 가리키고 있는 WSDL 서비스를 향해 서비스 요청한다. 이것은 Dynamic Invocation Interface(DII)을 잘 알려져 있다. DII를 이용하여 특정포트 ,특정 서비스명을 통해 서비스 이용이 가능하다. web services 를 위한 reflection 에 대해 생각해보라. 아래 클라이언트 코드를 보여주고 있다.

```
package org.jboss.ws.client;
```

```
import org.jboss.ws.hello.Hello;

import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.Call;

import javax.xml.namespace.QName;

import java.net.URL;

public class HelloClientDII
{
    public static void main(String[] args)
        throws Exception
    {
        String urlstr   = args[0];
        String argument = args[1];

        System.out.println("Contacting webservice at " + urlstr);

        URL url =  new URL(urlstr);

        String ns        = "http://hello.ws.jboss.org/";
        QName  qname     = new QName(ns, "HelloService");
        QName  port      = new QName(ns, "HelloPort");
        QName  operation = new QName(ns, "hello");

        ServiceFactory factory = ServiceFactory.newInstance();
        Service        service = factory.createService(url, qname);
        Call           call    = service.createCall(port, operation);

        System.out.println("hello.hello(" + argument + ")");
        System.out.println("output:" + call.invoke(new Object[] {argument}));
    }
}
```

The following two commands can be used to run the DII client against both the JSE and EJB web services we have created.

```
[examples]$ ant -Dchap=ws -Dex=1b run-example
```

```
[examples]$ ant -Dchap=ws -Dex=2b run-example
```

12.3.2. Service references

The JAX-RPC examples in Section 12.3.1, "A JAX-RPC client" all required manual configuration of the WSDL URL and knowledge of the XML nature of the web services in question. This can be a configuration nightmare, but if your code is a J2EE component there is another option. J2EE components can declare service references and look up preconfigured `Service` objects in JNDI without needing to hardcode any web service references in the code.

JAX-RPC는 12.3.1 절 "A JAX-RPC client" 에서 WSDL URL수작업과 web services 에 대한 xml 에 대해 필요한 부분을 다루었다. 이러한 설정은 악몽 같을 수 있다. 그러나 여러분의 코드가 J2EE component 이라면 여기서는 또다른 옵션이 된다. J2EE components 는 service references로 선언할 수 있다. 그렇게 하면 코드내에 어떠한 web service references 를 하드코딩 없이 JNDI 내의 선설정된 Service objects 를 look up 할 수 있다.

이 작업을 어떻게 하는 지 아래 코드에서 보여주고 있다. hello web service 를 호출하는 부분을 잘 보기 바란다.

```
package org.jboss.ws.example;

import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;

import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceException;

import org.jboss.ws.hello.Hello;

public class ExampleBean
    implements SessionBean
{
    public String doWork()
    {
        try {
            Context ctx     = new InitialContext();

            Service service = (Service) ctx.lookup("java:comp/env/services/hello");
            Hello   hello   = (Hello)   service.getPort(Hello.class);

            return hello.hello("example bean");
        } catch (NamingException e) {
```

```
            throw new EJBException(e);

        } catch (ServiceException e) {
            throw new EJBException(e);

        } catch (RemoteException e) {
            throw new EJBException(e);
        }
    }

    public void ejbCreate() {};
    public void ejbRemove() {};

    public void ejbActivate() {}
    public void ejbPassivate() {}

    public void setSessionContext(SessionContext ctx) {}
}
```

ExampleBean 은 doWork 메소드를 통해 hello web service 를 호출하고 있다. 여기서 dyamic proxy invocation 메소드가 사용되었다.
여기서 흥미로운 것은 ENC 에서 JNDI lookup 을 통해 service reference를 얻어 온다는 것이다.

Web service references 는 **ejb-jar.xml** 파일의 service-ref element 를 이용하여 선언되어 있다.

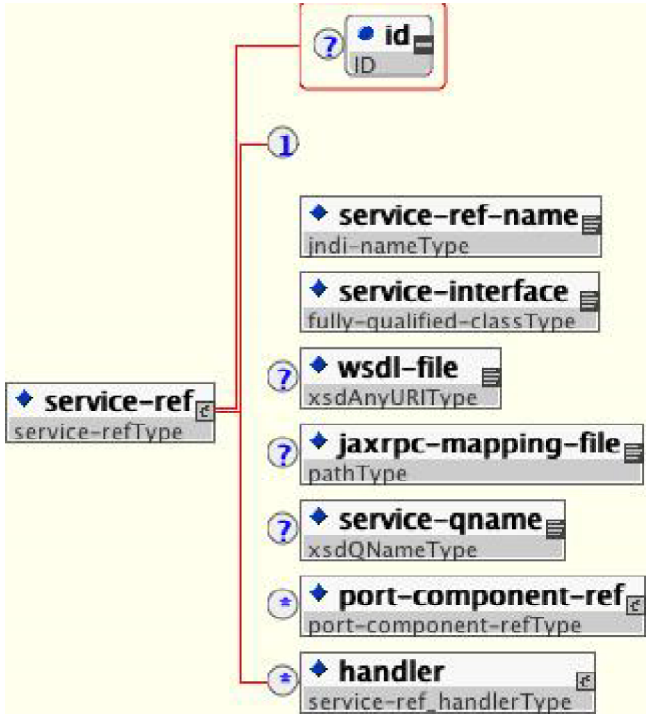

Figure 12.3. The service-ref content model

service-ref 에 대한 elements 의 설명이다.

- **service-ref-name**: This is the JNDI name that the service object will be bound under in the bean's ENC. It is relative to `java:comp/env/`.
- **service-interface**: This is the name of JAX-RPC service interface the client will use. Normally this is `javax.xml.rpc.Service`, but it's possible to provide your own service class.
- **wsdl-file**: This is the location of the WSDL file. The WSDL file should be under `META-INF/wsdl`.
- **jaxrpc-mapping-file**: This is the location of the JAX-RPC mapping file. It must be under the `META-INF` directory.
- **service-qname**: This element specifies the name of the service in the web services file. It is only mandatory if the WSDL file defines multiple services. The value must by a QName, which means it needs to be a namespace qualified value such

as ns:ServiceName where ns is an XML namespace valid at the scope of the service-qname element.

- **port-component-ref**: This element provides the mapping between a service endpoint interface and a port in a web service.
- **handler**: This allows the specification of handlers, which act like filters or interceptors on the current request or response.

service-ref 에 Example session bean 에 hello web service를 선언하는 예를 아래 보여주고 있다.

```
<session>
    <ejb-name>Example</ejb-name>
    <home>org.jboss.ws.example.ExampleHome</home>
    <remote>org.jboss.ws.example.Example</remote>
    <ejb-class>org.jboss.ws.example.ExampleBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <service-ref>
        <service-ref-name>services/hello</service-ref-name>
        <service-interface>javax.xml.rpc.Service</service-interface>
        <wsdl-file>META-INF/wsdl/hello.wsdl</wsdl-file>
        <jaxrpc-mapping-file>META-INF/mapping.xml</jaxrpc-mapping-file>
        <service-qname xmlns:hello="http://hello.ws.jboss.org">
            hello:HelloService
        </service-qname>
    </service-ref>
</session>
```

EJB deployer 는 java:comp/env/service/hello 명 아래 JNDI 의 bean 에 대해 사용가능한 Service Object 를 만들어 준다. Service Object는 hello web service 에 해당된다. session bean 은 보통의 web services operation s 을 호출 하는 것과 같다.

web services 설정 옵션의 대부분은 표준이다. 따라서 여기에 크게 깊게 다룰 필요성은 없다. 그러나 **JBoss 는 jboss.xml deployment descriptor 의 service-ref 를 통해 추가적인 web services 설정 옵션을 제공**하고 있다. service-ref elements 에대한 모양은 아래에서 보여주고 있다.

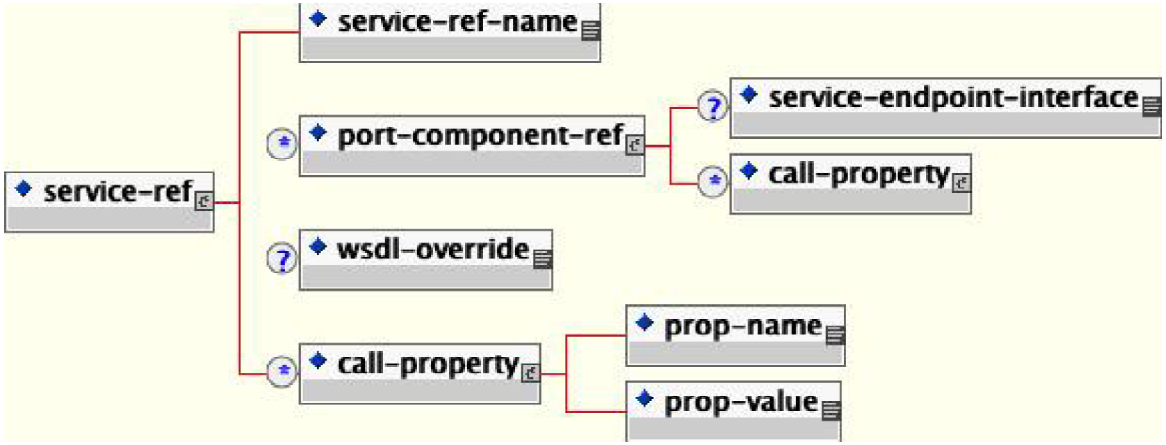

Figure 12.4. The jboss.xml service-ref content model

설정 elements 에 대해 아래에서 설명하고 있다.

- **service-ref-name**: This element should match the `service-ref-name` in the `ejb-jar.xml` file that is being configured.
- **port-component-ref**: The `port-component-ref` element provides additional information for a specific port. This includes properties that should be associated with the JAX-RPC stub for the port.
- **wsdl-override**: This provides an alternate location for the WSDL file. The value can be any valid URL. This can be used in co-ordination with the `wsdl-publish-location` to get the final WSDL file for a locally published web service. It could also be the URL of a remotely published WSDL that you don't want duplicated in the deployment file.
- **call-property**: This sets properties on the JAX-RPC stub.

WSDL 이 wscompile을 통해 생성되지만 SOAP address 를 포함하지는 않는다. JBoss 의 **jboss.xml** 파일에서 서비스의 wsdl url 이 override 되게 된다.
아래 예를 보자

```
<!DOCTYPE jboss PUBLIC
          "-//JBoss//DTD JBOSS 4.0//EN"
          "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
    <enterprise-beans>
        <session>
```

```
        <session>
            <ejb-name>Example</ejb-name>
            <service-ref>
                <service-ref-name>services/hello</service-ref-name>
                <wsdl-override>http://localhost:8080/hello-servlet/Hello?wsdl</wsdl-override>
            </service-ref>
        </session>
    </enterprise-beans>
</jboss>
```

샘플을 실행하기 위해선 아래와 같이 한다.

```
[examples]$ ant -Dchap=ws -Dex=3 run-example
...
run-example3:
     [echo] Waiting for 5 seconds for deploy...
     [copy] Copying 1 file to /tmp/jboss-4.0.3/server/default/deploy
     [echo] Waiting for 5 seconds for deploy...
     [java] output:Hello example bean!
```

service-ref element 는 ejb-jar.xml 파일에 제한적이지 않다. J2EE component 어디서든 가능하다. service reference는 web.xml 파일에서도 설정 가능 하다. 또는 application-client.xml 파일에서도 가능하다.
